2017

Maximizing Cache Value For Distributing Content Via Small Cells In 5G

Ibrahim Khalaf Freewan
University of Mississippi

MAXIMIZING CACHE VALUE FOR DISTRIBUTING CONTENT VIA SMALL CELLS IN 5G

A Thesis
presented in partial fulfillment of requirements
for the degree of Master
in the Engineering Science with Emphasis in Telecommunications
The University of Mississippi

by

Ibrahim Freewan

May 2018

ABSTRACT

The objective of this thesis is to develop an algorithm for distributing content at small cells in 5G, wherein small cells (SCs) are set up instead of macro cells to serve mobile users. Users may simultaneously access between one and some maximum number of SCs, the actual number of SCs being drawn from a distribution. The source library is located away from these sites, and it stores files as a collection of RaptorQ-encoded symbols. By using RaptorQ symbols a very large number of distinct encoding symbols can be generated and a collection of received encoded symbols slightly larger than the number of source symbols can be used to recover the original file with linear time complexity. A number of these encoded symbols should be placed in SC caches to facilitate efficient download. The main objective of this thesis is to develop an algorithm of low complexity to determine the number of encoded symbols of each file that should be cached at each SCs as a function of cache size. Each file is characterized by the number of encoded symbols required to reconstruct the file at the user equipment and its download preference probability.

The optimization problem considered is to determine the symbols distribution for a set of files stored in the library in order to minimize backhaul. The objective function is the average value of storing a set of encoded symbols per file download, constrained by available cache memory. Parameters are the files' preference probabilities and sizes, coverage areas probabilities, the total number of files in the library, and the cache capacity.

This study contributes to the literature by developing an $n \log n$ algorithm to solve the optimization problem, extending previous results from constant files size for all files to arbitrary actual files sizes for all files, and extending distribution portions from continuous fractions of files to integer number of symbols.

DEDICATION

To my Mom's soul.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Prof. John Daigle for the continuous support of my graduate study at the University of Mississippi. My grateful appreciations also go to Prof. Feng Wang for the many helpful comments and suggestions on my research. I would like to thank my professors, teachers, colleagues, and family members, whom supported me in all aspects of life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

The objective of this thesis is to develop a low-complexity approach to maximizing the value of cache located at small cells of cellular networks. In general, small cells communicate directly with users to deliver content from the network, and cache at the small cells is used to reduce the amount of downloading from the network that is needed to satisfy demand.

For the specific case under consideration in this thesis, the content to be delivered is a file from a library of files. Each file is stored as a collection of encoded symbols which can be generated from equal sized fragments of the file, called source symbols, using the RaptorQ encoding technique. In order to download a file, the user first downloads a prescribed number of distinct encoded symbols from the network and then recovers the original file using RaptorQ decoding. The reason for using RaptorQ is that the nature of the problem in some cases requires that the number of distinct encoded symbols placed at each cell are greater than the number of source symbols themselves. In addition, RaptorQ is an on the fly encoding and decoding technique that has computation complexity that is linear in file sizes.

The idea of caching is that a subset of the encoded symbols of the files are stored in the small cells so that downloading of a given set of symbols from the library is not required every time a given file is downloaded. In order to maximize the value of the cache by minimizing download, symbols that are needed the most should be stored in the cache to the extent possible. Given a specific cache size and a specific collection of files, the objective of this thesis is to develop a low-complexity algorithm to compute the number of symbols from each file that should be placed in the cache of each small cell in order to minimize the download from the network, which is called *backhaul*.

The developed algorithm takes into account several variables that play a big role in determining the number of encoded symbols from each file placed in the cache: cache capacity at the small cell, file sizes and their preferences, number of files in the library, and network topology. A large size of cache allows us to store more content. Therefore, it reduces the amount of backhauled download. File sizes and their preferences directly affect the number of encoded symbols that should be delivered to the user and how many times these symbols would be downloaded if they were not cached, respectively. Network topology determines the number of small cells with which a user can communicate with. The user who has access to multiple small cells requires a lower number of cached symbols at each cell than the user who has only access to one small cell.

The objective is to develop an algorithm considering all variables above in addition to low time complexity. The complexity is a function of the number of files that are stored in the library. In general, this number is huge. Therefore, a low time complexity algorithm is important. On the other hand, the problem would be very simple if the users do not have multiple simultaneous access to small cells. In this case, the solution would prioritize the most popular content regardless any other variables mentioned above. In contrast, the problem will be NP-hard if the file stored in the cache is restricted to store only complete files [4].

In this study, the optimization problem is transformed into a list sorting problem with consideration of all variables that are just discussed. To the best of our knowledge, no study has achieved the time complexity and the optimal result obtained by the algorithm presented herein. For example, of the interesting results obtained is that the time to determine the number of encoded symbols from each file among a set of 1 million files that is stored in a library is approximately 80 seconds on Intel ®Core™i5 laptop, whereas methods found in the literature for solving such a problem required about 80 seconds of computations for a set of only a few hundreds files. In addition, our results show the significant affect of caching content as RaptorQ encoded symbols in reducing the amount of backhaul, where storing 10%

of trending YouTube videos of 3 GB size is enough to reduce the data rate on the backhaul by 65% in one scenario.

The structure of this thesis is organized as follows. The next chapter discusses content caching and delivery network infrastructure. The general architecture for small cell networks in the urban environment is presented together with a discussion on the architecture aspects for supporting content caching and delivery service. Related work is presented in Chapter 3. A discussion on RaptorQ is presented in Chapter 4. Chapter 5 presents the system model. The formulation of the problem and low time complexity algorithm that solves the problem optimally are presented in Chapter 6. Extensive numerical results showing the low time complexity of the developed algorithm and the effectiveness of the cache capacity in reducing backhaul usage are addressed in Chapter 7. Conclusions are drawn in Chapter 8.

CHAPTER 2

Content Caching and Delivery Infrastructure

In this chapter, we discuss the infrastructure for content caching and delivery network service. The next section presents the general architecture for the urban small cells network (SCN) with focusing on the part that is related to our system model, that is, small cell radio access netwrok (SC RAN), where the architecture itself is proposed by the small cells forums (SCF) organization. Section 2.2 addresses the architecture aspects for the content caching and delivery service where the related framework for supporting this service is presented. The matiral of this chapter is referred to white papers for SCF orgnization [1], [2].

2.1 Urban SCN End-to-End Architectural Framework

The general architectural framework for urban SCNs is shown in Fig. 2.1. The dotted line indicates optional element in the architecture. For example, SC access point may be directly connected to SeGW or through anchor access point. Boxes do not mean necessarily physical nodes, they may be logical functional nodes. In practice, the physical node may implement one or more logical functional nodes. As shown in the figure, there is a number of architectural domains in the architecture, namely, SC RAN, macro RAN, backhaul transport network and operator core network (operator CN).

The SC RAN is comprised of a number of SC access points. In practice, SC AP may be 3G or LTE. The interface between SC access points is Iurh for 3G and X2 for LTE. It also has different classes such as home, local area and medium range. In addition, The access points are deployed as an isolated single point that directly connects to the operator CN or they may be developed within a cluster. In the case of cluster, the access point connects to anchor SC that provides the connectivity between the access point and the operator CN in

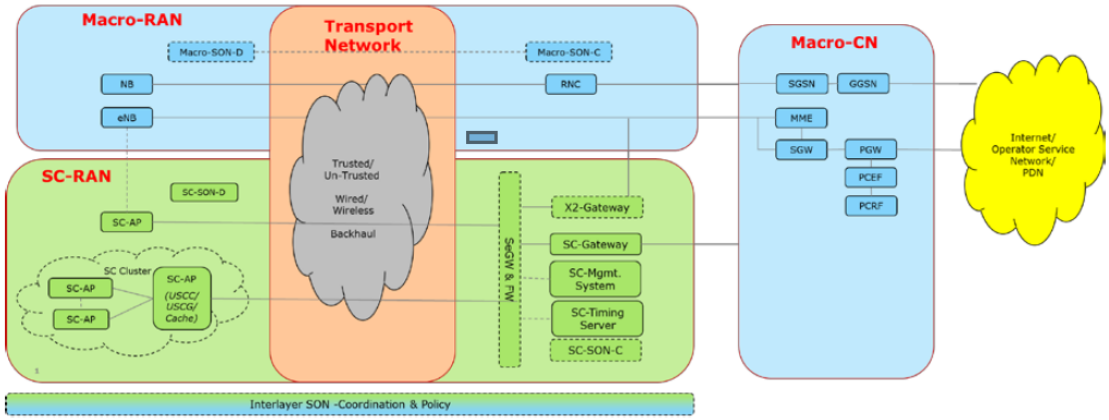

Figure 2.1. Generic urban small sells network end-to-end architecture framework. [Fig. 2-1, SCF 088]

addition to local gateway functionality. The links that connect all SC access points to the operators CN is a part of backhual transport network. Depending on the security measures that need to be taken, the backhaul transport network may be either trusted or untrusted. In addition, both wired and wireless physical layer technologies are possible. Moreover, SC RAN consists of several fnuctionalites such as SC managements system, X2 gateway, timing sever and self organizing networks (SON) functions. Signaling between the SC access point and SC RAN functionalities bypasses the SeGW but be secured via transport level security schemes.

2.2 Architecture Aspects to Support Content Services

Urban small cells platform offers the content management service where the traffic localization that is achieved via content caching and delivery in the small cell networks can benefit the operators in several ways, including: reducing load on the backhaul network and the core network, improving overall user experience and generating new revenue opportunities for SCN owners.

There are two distinct concepts related to caching in urban SCNs: local content and cached content. Local content refers to content originally stored in the local network, that is, it is locally available. In contrast, cached content is the content that is originally located

elsewhere in the network, that is, it is accessible via the public Internet, or a copy of the original content is stored in SC cache. In general, the content might be web accessible advertisements, videos and web pages. The content provider prefer to cache the content in the small cells network in order to provide more efficient access in terms of latency to the users and offload traffic from the backhaul and CN.

2.2.1 Content Delivery and Caching Architecture Framework

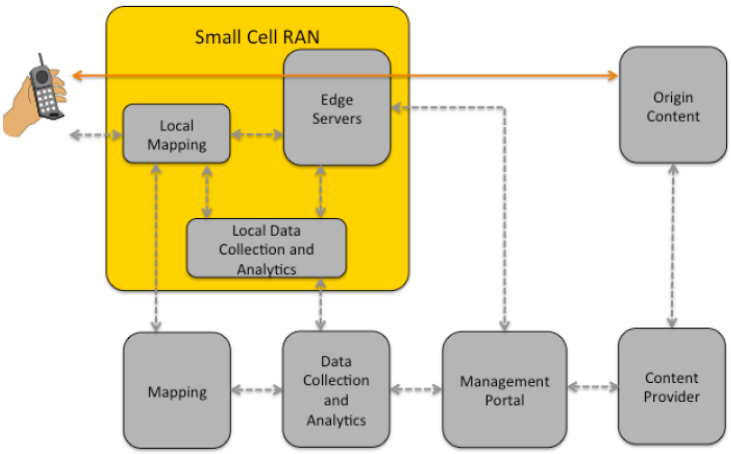

Figure 2.2. An example architectural framework for caching at small cells. [Fig. 2-1, SCF 093]

Small cell forums origination proposed an architecture framework for supporting content delivery network service. The architecture framework, as shown in Fig. 2.2, presents functions that should be included in the SC RAN to support this service. These functions are: local mapping, local data collection and analytics, and edge servers where the content itself is stored. They may also be deployed in three different places: in the small cell itself, in the enterprise SC concentrator or on the edge of the SC networks (SC gateway). On the other hand, an appropriate APIs are installed in the core network to support the corresponding function. For example, centralized mapping, data collection and analytics, and management portal services.

Mapping services is an important service to identify the request from user equipment (UE) for a piece content locally, and subsequently, directs the UE request to the local content. There are several possible approaches for mapping. These approaches can be organized into two groups, namely, explicit and transparent. Explicit methods (directed methods) redirect the UE to the content that is stored locally. In contrast, transparent methods forward the user request for content to where the content stored, that is, the edge server, without indicting to the user that the request has been served locally from the cache.

DNS proxy and HTTP servers are two examples of an explicit method for implementing the local mapping service. When the content is available, the proxy server can identify the request content and resolve the IP address of an edge server, whereas the HTTP server can identify the requested content by an HTTP re-direction response that directs the UE to the edge server where the content is cached. The HTTP server, identified in the re-direct response, may be co-located with the origin server, implemented in a mobile gateway, or located in the small cell network. An example of a transparent method is by using a layer 4 (TCP) based redirect mechanism where the mapping server would autonomously forward requests that can be serviced by edge servers to those edge servers. Mixed content mapping methods is also possible to be employed in the small cell network. For example, the advertisement that is embedded in a web page may be cached locally, while the web page itself is not locally cached.

The data collection and analytics is used to collect statistics related to the content that can benefit from being cached and how often local content is accessed. For example, hit and miss rates could provide the content provider enough information about the files preferences, which is considered very important for any suggested algorithm for solving content placement problem. In Chapter 6, we will see how the file preferences present in the formulation of the problem.

The management portal services allow the origin content provider and content publisher to manage any content that is stored locally, and to view analytics, or statistics, about

how such content is accessed. It is corresponding to content placement methods. In Chapter 6, we will present maximal incremental gain algorithm that solves the problem in optimal way for the system model that is presented in Chapter 5.

Management portal services may be transparent or explicit. Explicit methods are used when the content owner either directly places copies of content in the local network for cached retrieval or the network does so under explicit direction from the origin content provider. Transparent methods are used when the local network autonomously decides to create a cached copy of content based on local rules or policies. These policies may or may not be influenced by the content owner. In general, transparent placement methods must carefully consider copyright issues. It may also require content owner's permission to copy the data. In practice, there are several ways to implement this permission, for example, using HTTP headers such as *max-age* and *no-cache*.

2.2.2 Mobility Aspects Related to Delivery of Local Content

The content provider must be aware that the cached content is accessible to user since the provider is able to redirect the user to cached content. For example, there is an issue when a user moves from small cell to another and the cached content is being served from an edge server that is colocated with the small cell itself. When the user connects to the small cell that has access to the cached content and then move to another small cell or macro cell that does not has access to the same cached content, in this case, the care must be taken to ensure the user can seamlessly transition to access the original content. If the user moves to small cell that has access to the same content, this would be less than an issue.

It is also imperative that uniform policy enforcement, across local and macro network, is implemented in case when a macro network operator provides access to local cache. For example, if the user has been prevented from accessing content in the macro network, then the local network should prevent the user from accessing a cached copy of the same content.

CHAPTER 3

Related Work

In this chapter, we discuss related work for the content placement problem with a focus on the studies that are presented in [3] and [4]. The problem is formulated in different ways in different models but the objective of both studies is to minimize the amount of downloading from the core network.

In [4], the authors suggest a novel approach that allows users to communicate with multiple small cells called helpers. These helpers are provided with cache memory to store the most popular files in order to assist a macro base station (MBS) handling a part of user requests. The requested files that can not be directly served from the helper caches must be downloaded from the MBS.

Files are assumed to be completely stored or not stored at all in helpers. Moreover, a given file might be cached at several neighboring helpers such that the user who has access to any one of a group can download a copy of the file from the cache of any of these helpers. All files are assumed to have the same size. The size equality constraint can be easily lifted by breaking larger files into small blocks of the same length and each of these blocks would have the same preference as the larger file. For example, given a large file that is fragmented into 4 small blocks. Then, each block among 4 blocks will be considered different file with the same preference as the large file. In addition, these different files are correlated; that is, users who download one of 4 files will download the other 3 files in order to have the complete large file. The popularity distribution of the files is assumed that it changes slowly and it can be learned by user requests history.

The study assumed that there is no delay when the file is delivered via helpers, whereas the delay is presented if the requested file is delivered by the MBS. The objective is

to determine which files should be stored at the helpers in order to minimize this delay. The delay is defined as the time required to deliver an arbitrary requested file of fixed size to the user via the MBS. That is, the delay is zero if the requested file is served by the helpers, and it is w_k if the file is delivered via MBS where k is the user index. The objective function of the optimization problem that is formulated in this study is given by

$$D = w_k \left(1 - \sum_{n \in A_k} p_n\right), \tag{3.1}$$

where A_k is the collection of complete files that are stored in the helpers and that user k can access, and p_n is the probability that the file n will be requested. The interpretation of $(1 - \sum_{n \in A_k} p_n)$ is the the probability that the user k requests some file and this file is not stored at the helpers neighboring the user.

The problem, defined as helper design problem (HDP), is proved to be NP-hard. Then, the study expresses the problem as a maximization of monotone submodular function over matriod constraints in order to take the advantage of the approximation results that are done in [7] for submodular functions. Matroid is structure that generalizes the concept of independence. For example, if we have a finite ground set, S, matriod then is a way to label subsets of S as independent. Submodular functions capture the concept of diminishing returns; when the set becomes larger, the benefit of adding a new element to the set will decrease. In HDP, the element in the subsets is the complete file that is stored in helpers, where the collection of cached complete files in each helper is a subset that is independent of other subsets. The collection of the subsets is the finite ground set, S.

The study suggested an algorithm that determines which completed files should be stored at the helpers. The developed algorithm is within a factor of 2 from the optimality; that is, the performance gain for the solution that is produced by algorithm is half of the performance of the optimal solution. The study did not clarify the time complexity of the developed algorithm. The algorithm also requires a central coordinator to collect information

of the whole network to find the solution. In addition, solution methodology does not account for the fact that a user may have simultaneous access to multiple helpers.

In [3], Bioglio *et al* defined the problem of caching MDS-encoded content at the wireless edge for a heterogeneous network, and derive the backhaul rate performance of a caching scheme based on storing MDS encoded symbols. MDS codes is stand for maximum distance separable codes. In practice, using MDS(a, b) codes can generate a encoded symbols from b source symbols, where source symbols is the equal sized fragments that each file is divided into. The user who receive any subset of b encoded symbols can reconstruct the file. The study also formulates the optimal caching scheme for MDS-encoded content as a convex optimization problem. Moreover, the optimal solution for the this convex problem as it is formulated outperforms any other caching schemes.

According to the system model of [3], users have simultaneous access to multiple small cells called small base stations (SBSs). These SBSs are provided with a cache memory and backhaul link that connects the SBS to MBS. The backhaul link is used to fetch the requested file if the related encoded symbols are not cached locally at the SBSs.

In the system model, files are originally stored in a library that is located away from SBS sites and stored as collection of MDS encoded symbols. Unlike the model of [4], fractions of files may be stored in the cache. The files are also assumed to be the same size, where this constraint can be lifted by fragmenting the larger file into small blocks of the same length. For example, if we have two files with equally likely to be requested by an arbitrary request. The size of the first file is 200 encoded symbols and the second one is 100 encoded symbols. If we fragment the first file into two subfiles, each subfile is with a size of 100 encoded symbols. Now, each 100 encoded symbols among three file, the two subfiles and the second file, will have the same preferences. As we will see in Chapter 6, the value of the cache when we add symbols from any two different files that have the same preference will be the same. Thus, the value of the cache from adding a certain number of encoded symbols from any of three files will be the same. Let us assume that adding the first 50 encoded symbols from the large

file makes the value of the cache is $2c$, where c is some value. Then, the first 25 encoded symbols from each of the two subfiles will make the value of the cache is c. More detail on this point will be in Chapter 6.

The objective of the optimization problem that is defined in this study is to determine the fraction of each file from the library that should be placed at the SBS caches in order to minimize the backhaul link usage. The average backhaul rate for an encoded caching placement scheme for the solution $(m_1, m_2, ..., m_N)$ is given by

$$R_{\text{MDS}} = \sum_{i=1}^{S} \sum_{j=1}^{N} \gamma_i p_j \left(1 - \min\left\{1, \frac{i m_j}{n}\right\}\right), \tag{3.2}$$

where i is the number of SBSs that the user can connect to, γ_i is the probability that user has access to i SBSs, N is the number of the files in the library, n is the number of the fragments that each file is fragmented to, m_j is the number of encoded symbols of the file j that is placed in the cache, and p_j is the file preference. In order to reconstruct the file at user equipment, n encoded symbols should be delivered to the user.

The formulation of the problem is done by assuming that a user has access to i SBSs and request some file, F_j. Each SBS of the i^{th} SBSs stores m_j of n encoded symbols at its cache. The number of symbols that will be then download by the backhaul link is given by

$$1 - \min\left\{1, \frac{m_j}{n}\right\}, \tag{3.3}$$

where the amount of download is normalized to the value of n. If the user has access to i SBSs with probability γ_i, the average amount of downloaded symbols is then given by

$$\sum_{i=1}^{S} \gamma_i \left(1 - \min\left\{1, \frac{m_j}{n}\right\}\right). \tag{3.4}$$

Formula 3.4 is for one file. If we have N files, the total amount of download on the backhaul

link for all files is given by

$$\sum_{j=1}^{N}\sum_{i=1}^{S} p_j\gamma_i \left(1-\min\left\{1,\frac{m_j}{n}\right\}\right). \tag{3.5}$$

On the other hand, the average backhaul rate for random caching placement scheme for the solution $(m_1, m_2, ..., m_N)$ is given by

$$R_C = \sum_{j=1}^{N}\sum_{i=1}^{S} p_j\gamma_i \left(1-\frac{m_j}{n}\right)^i. \tag{3.6}$$

The formulation of R_C is the same way as R_{MDS} except that (3.3) will be

$$\left(1-\frac{m_j}{n}\right)^i. \tag{3.7}$$

Formula (3.7) is corresponding to the probability that m_j fragments of file j are not replicated at other SBS among the i^{th} SBSs that a user has access. The replication of fragments is referred to that we do not use MDS codes. Thus, we could not have distinct m_j fragments of the file for each SBS among the i^{th} SBSs like the case of storing the file as MDS encoded symbols.

By comparing the average data rate of each schemes, R_{MDS} and R_C, the authors proved that for any solution $(m_1, m_2, ..., m_N)$, it holds that $R_{\text{MDS}} \leq R_C$. The difference between (3.2) and (3.6) is $1-\min\left\{1, i\frac{m_j}{n}\right\}$ and $\left(1-\frac{m_j}{n}\right)^i$, respectively. In order to prove $R_{\text{MDS}} \leq R_C$, we then need to prove

$$1-\min\left\{1, i\frac{m_j}{n}\right\} \leq \left(1-\frac{m_j}{n}\right)^i. \tag{3.8}$$

Since $i \geq 1$ and $0 \leq m_j/n \leq 1$, from Bernoulli's inequality

$$1-i\frac{m_j}{n} \leq \left(1-\frac{m_j}{n}\right)^i. \tag{3.9}$$

Since $0 \leq \left(1 - \frac{m_j}{n}\right)^i$, it is true that

$$1 - \min\left\{1, i\frac{m_j}{n}\right\} = \min\left\{0, 1 - i\frac{m_j}{n}\right\} \leq \left(1 - \frac{m_j}{n}\right)^i; \tag{3.10}$$

that is,

$$1 - \min\left\{1, i\frac{m_j}{n}\right\} \leq \left(1 - \frac{m_j}{n}\right)^i. \tag{3.11}$$

In this thesis, the problem is formulated as below

Problem: Symbols Placement

$$\max_{m_1, m_2, ..., m_N} R(m_1, m_2, ..., m_N) = \sum_{j=1}^{N} p_j \sum_{i=1}^{S} \gamma_i \min\{E_j, im_j\} \ . \tag{3.12}$$

Subject to:

$$\sum_{j=1}^{N} m_j = M_C \tag{3.13}$$

$$0 \leq m_j \leq E_j \qquad m_j \in \mathbb{Z} \quad \forall j \in \{1, 2, ..., N\}, \tag{3.14}$$

where N is the number of files that are originally stored elsewhere in the network, M_C is the cache capacity, m_j is the number of encoded symbols of the file F_j that stored at SC and E_j is the minimum number of encoded symbols that is needed to reconstruct the all source symbols that file F_j consists of. We will address this point in the next chapter throughout a discussion on RaptorQ. The value m_j is restricted to an integer up to E_j. The value γ_i is the probability that a user has simultaneous access to i small cells, and S is the maximum number of small cells that user can connect to. The interpretation of the objective function is that the value of the SC cache is maximized by minimizing the amount of download from the network. Chapter 6 will address the problem formulation and the analysis that leads us to solve the problem optimally with low time complexity algorithm.

CHAPTER 4

RaptorQ

RaptorQ is forward error correction technology, which is fully specified in RFC 6330 [6]. To use RaptorQ, a unit of source data is broken into a number of blocks, and each block is encoded independently of all other blocks. Each block is divided into a number of K equal sized fragments called source symbols. In addition, each block is augmented with $K' - K$ zero padding symbols, where K' must be chosen from Table 2 in RFC 6330 as the smallest value that is greater than K; that is, the block is extended to K' source symbols.

The next step is to generate $L = K' + S + H$ intermediate symbols where S and H are fractions of K'. A total of H HDPC are generated by performing a linear combination on a large fraction of source symbols, and S LDPC symbols are generated by performing a linear combinations on a small fraction of the source symbols. Both S low density parity check and H high density parity check symbols result in coded symbols.

The basic idea of encoder is that we generate encoded symbols from the intermediate symbols as a collection of source symbols and repairs symbols using the same process. Each encoded symbol is a linear combination of a subset of intermediate symbols. Each encoded symbol has an internal symbols ID (ISI) that uniquely identifies the intermediate symbols associated with each encoded symbol for encoding and decoding purpose. The values of the ISI for the extended source symbols are between 0 and $K' - 1$ and the values for the repair symbols are $K', K' + 1, ...$

The intermediate symbols can be grouped into W Luby transform (LT) symbols and P permanently inactive (PI) symbols, where $L = W + P$. The W LT symbols then consist of S symbols and $B = W - S$ of K' intermediate symbols. Similarly, the P PI symbols consist

of H HDPC symbols and $U = P - H$ of the other K' intermediate symbols. The values of S, H and W are determined for a specific value of K' from Table 2 mentioned in RFC 6330.

Let us now denote the intermediate symbols by $C = ([C[0], ..., C[L-1])$. Then, $C_1 = (C[0], ..., C[B-1])$ are symbols that are LT symbols but not LDPC symbols. The symbols $C_2 = (C[B], ..., C[B+S-1])$ are S LDPC symbols. The next subset of symbols, $C_3 = (C[W], ..., C[W+U-1])$, are the PI symbols but not HDPC symbols. The last symbols, $C_4 = (C[L-H], ..., C[L-1])$, are H HDPC symbols. Matrix A is then defined in $A \times C = D$, where D is column vector consist of $S+H$ zero symbols and K' source symbols. The structure of matrix A and C are shown in Fig. 4.1.

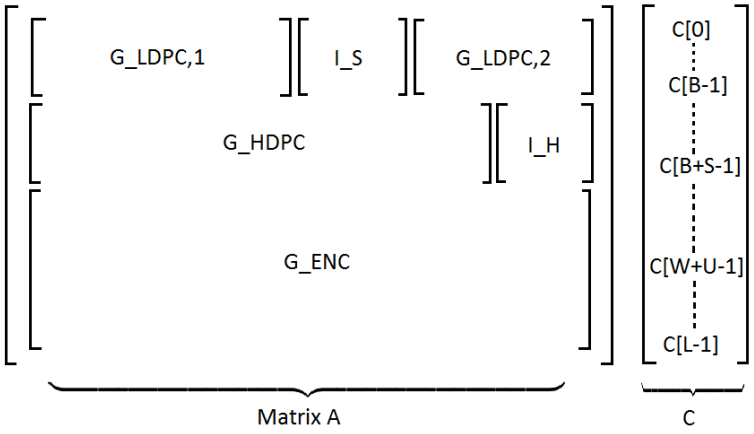

Figure 4.1. Matrix A and C at the encoder side.

Each submatrix in matrix A is evaluated as shown in Table 4.1, where T denotes transpose operation and C' is $([C'[0], ..., C'[k'-1])$, the K' source symbols.

For each repair symbol that will be generated from the intermediate symbols, tuple of (d, a, b, d_1, a_2, b_1) is generated for a specific ISI value using a tuple generator, where the tuple parameters are used by the encoder to determine the number and the set of intermediate symbols that will be combined to form the encoded symbol. The repair symbols are then generated by first have the extended source symbols and then generating the L intermediate symbols as explained above. After this, we generate a tuple based on the ISI to determine

which subset among intermediate symbols contributing in the encoded symbol. The encoded symbol value then is the result of the linear combination of intermediate symbols.

Table 4.1. Definition of each submatrix in A matrix.

Submatrix	size	Evaluated by
G_LDPC,1	$S \times B$	G_LDPC,1 $\times C_1^T$ + G_LDPC,2 $\times C_3^T + C_2^T = 0$
G_LDPC,2	$S \times P$	
G_HDPC	$H \times (K' + s)$	G_HDPC $\times (C[0], ..., C[L - H - 1])^T = C_4^T$
G_ENC	$K' \times L$	G_ENC $\times C^T = C'^T$
I_S	$S \times S$	Identity matrix
I_H	$H \times H$	Identity matrix

At the decoder side, any large set of distinct encoded symbols, both source and repair symbols, allows us to recover the intermediate symbols no matter which specific encoded symbols are received. Once we received the encoded symbols the value of D in $A \times C = D$ is obtained. The matrix A can be built from the encoded symbols, where the number of the rows in A is $S + H + K' - K$ plus the number of received encoded symbols, which is greater than K' and typically greater than K . The value of A is not necessarily the same as matrix A at the encoder. It depends on the received encoded symbols. On the other hand, the first $S + H$ rows still be the same as the corresponding rows in matrix A at the encoder and the rows corresponding to padding symbols will be the same. Here, the C is the only the unknown variable in the equation. By solving the equation using an efficient RaptorQ decoder, the intermediate symbols can be recovered. In case of missing source symbols, they can be obtained from the intermediate symbols using the A matrix that was originally used to find the intermediate symbols. More specifically, if the source symbols with ISI j is missing then the vector of intermediate symbols is multiplied by the $S + H + j^{th}$ row of the original A matrix to yield the j^{th} source symbol.

CHAPTER 5

System Model

Fig. 5.1 shows a diagram of dense cellular network. Small dash circles represent coverage's SCs and the large solid circle represents a macro base station. Yellow marks are users that have simultaneous access to S SCs at maximum, where S is determined by the distribution of SCs and the probability that user has access to i SCs is denoted by γ_i. The library of content F is located away from SC sites. The library stores N files, $\{F_1, F_2, ..., F_N\}$, as collections of encoded symbols, where N is the number of files. File F_j, $j = 1, 2..., N$, has size E_j measured in terms of the number of independent encoded symbols required to reconstruct the file at user equipment (UE), the vector $p = (p_1, p_2, ..., p_N)$ represents the probabilities that each file will be requested on an arbitrary requested; that is, the size of file F_j is E_j and the probability that it is requested on an arbitrary requested is p_j.

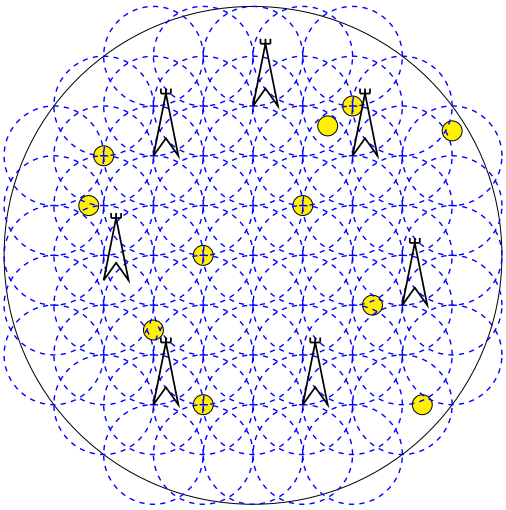

Figure 5.1. Diagram of dense cellular network.

Each SC is provided with a cache capacity, M_C, to store symbols. In addition, each SC has a backhaul link to connect to a mobile switching center (MSC), which in turn connects

to the network. The library is located elsewhere in the network. Users send requests to the i^{th} SCs to download some file stored in the library, and these SCs are responsible for delivery of the file to the users. SCs download available encoded symbols from their caches, and if the number of available symbols is insufficient for reconstruction, they use backhaul links to fetch the remaining symbols from the library and deliver them to the user.

The optimization problem is to determine the optimal set of encoded symbols among all files that minimize the average number of backhualed encoded symbols, which maximizes the average value of the cache at the SCs.

In this thesis, we focus on formulating and solving the optimization. We will discuss a few points describing how the system works. In practice, we need to determine how to serve an arbitrary request; that is, we need to determine, after calculating the optimal set of encoded symbols among all files, how many encoded symbols are available at the caches for this specific request that is sent by an user who has access to a certain number of SCs. For example, given the optimal set of encoded symbols, the number of encoded symbols that should be downloaded for a request that is sent by user who has access to four SCs might be different from the case where the user has access to one SC. This issue is a part of management portal function responsibilities as described in Chapter 2.

In this study, the file preferences is calculated by the number of requests for some file divided by the total number of requests for all files in the library . In practice, we need to develop an algorithm to track these requests. This algorithm may take the advantage of data collection and analytic function as described in Chapter 2. Another option is to consider the most popular contents in the the most popular website, for example, Facebook, Tweeter, and YouTube.

User mobility should also be counted in the calculations. In this study, we limit our work to the case where the user is stationary during the download period. That is, the SCs that serve the user simultaneously are the same throughout the download period. This assumption is reasonable since the data rate is expected to be very high in 5G. Thus, the

download period will be small enough to ensure that the user's access to the SCs does not change throughout the download period.

We also assumed that the backhaul link and user-SC link are error-free links. In addition, the encoded symbols are assumed to be one kB to simplify the analysis. In practice, the encoded symbol size should fit with the packet size of the wireless protocol that is used in 5G, GTP-U protocol. These assumptions have no effect on the correctness of the solution.

CHAPTER 6

Formulation of the Problem

In this chapter, we will formulate the optimization problem of symbols placement. Section 6.2 shows the analysis for the problem that leads us to solve the problem optimally with low time complexity algorithm. The algorithm, maximum incremental gain (MIG), is discussed in section 6.3.

6.1 Introduction

The symbols placement problem is defined as the optimization problem whose objective is to determine the number of encoded symbols of each file that should be stored at each SC. In order to maximize the average number of symbols available from cache per downloaded file, these SCs are set up in away that allow the user to have simultaneous access to multiple SCs. Let us assume that the user u sends a request to download some file located elsewhere in the network. User u has simultaneous access to i SCs. When the SCs receive the user request, they determine whether the number of encoded symbols stored at their caches are less than E_j or not. In case the number of encoded symbols is less than E_j, the SCs use the backhaul links to download the remaining symbols that are needed to reconstruct the file at user equipment. The number of encoded symbols that will be downloaded by these SCs via backhauls link is given by

$$E_j - im_j. \tag{6.1}$$

In case $im_j > E_j$, we then need to restrict the value of im_j to the file size, E_j. The number of encoded symbol to download via backhaul will be

$$E_j - \min\{E_j, im_j\}. \tag{6.2}$$

The probability that user can contact with i SCs is denoted by γ_i. If S is the maximum number of SCs that user might be connect to, the average number of the encoded symbols that need to be downloaded will then be

$$\sum_{i=1}^{S} \gamma_i \left(E_j - \min\{E_j, im_j\}\right). \tag{6.3}$$

We will show in a while how easily we can calculate the value of γ_i.

File j in the library has some probability to be requested at an arbitrary request. The probability is denoted by p_j. If we have N files in the library, the total average number of encoded symbols that need to be downloaded will be

$$\sum_{j=1}^{N} p_j \sum_{i=1}^{S} \gamma_i \left(E_j - \min\{E_j, im_j\}\right) \tag{6.4}$$

or, equivalently

$$\sum_{j=1}^{N} p_j \left(\sum_{i=1}^{S} \gamma_i E_j - \sum_{i=1}^{S} \gamma_i \min\{E_j, im_j\}\right). \tag{6.5}$$

Since $\sum_{i=1}^{S} \gamma_i E_j = E_j$, formula (6.5) reduces to

$$\sum_{j=1}^{N} p_j \left(E_j - \sum_{i=1}^{S} \gamma_i \min\{E_j, im_j\}\right). \tag{6.6}$$

The goal of the symbols placement problem is to minimize the usage of the backhaul. Therefore, the optimization problem is to minimize formula (6.6) over the variable m_j, that is, we need to minimize the number of encode symbols that will be downloaded by the backhaul links. The optimization problem can be written as

$$\min_{m_1, m_2, \ldots, m_N} \quad \sum_{j=1}^{N} p_j \left(E_j - \sum_{i=1}^{S} \gamma_i \min\{E_j, im_j\}\right). \tag{6.7}$$

The formulation so far is the same as that of [3] except that we extend the formulation to have arbitrary file sizes as improved to fixed file sizes. If we enter the term of $\sum_{j=1}^{N} p_j E_j$

into the parentheses and multiply by minus, we can take the constant $\sum_{j=1}^{N} p_j E_j$ out of the objective function. The optimization problem can be then expressed as

Problem: Symbols Placement

$$\max_{m_1,m_2,...,m_N} R(m_1, m_2, ..., m_N) = \sum_{j=1}^{N} p_j \sum_{i=1}^{S} \gamma_i \min\{E_j, im_j\} \ . \tag{6.8}$$

Subject to:

$$\sum_{j=1}^{N} m_j = M_C \tag{6.9}$$

$$0 \leq m_j \leq E_j \qquad m_j \in \mathbb{Z} \quad \forall j \in \{1, 2, ..., N\}, \tag{6.10}$$

where N is the number of files, M_C is the cache capacity, and m_j is the number of encoded symbol of the file F_j stored at the SC. The value m_j is restricted to an integer up to the minimum number of encoded symbols required to reconstruct all source symbols of the original file, E_j. The integer restriction is due to that the fact the encoded symbol can not be fragmented. The physical meaning of objective function is that we need to maximize the value of storing the encoding symbols at the SC cache. On the other hand, the physical meaning of the objective function in (6.7) is to minimize the traffic on the backhaul link.

The value of γ_i is the probability that user can connect to i SCs. Due to the overlapping between SCs, the coverage area of the small cell is divided into different subareas. Each subarea allows a user to communicate with a certain number of SCs. Let us denote the subarea size where user has access to i SCs by A_i, and user density at the subarea A_i by ρ_i. The probability that user has access to i SC is given by

$$\gamma_i = \frac{\rho_i A_i}{\sum_{i=1}^{S} \rho_i A_i}. \tag{6.11}$$

6.2 Analysis

The encoded symbols that are stored in the library are generated using RaptorQ technique. The concept behind RaptorQ is that given a file fragmenting into small fragments called source symbols, we can generate as many encoded symbols as needed on-the-fly. The user who receives any subset of size E_j of these encoded symbols can reconstruct the original file F_j.

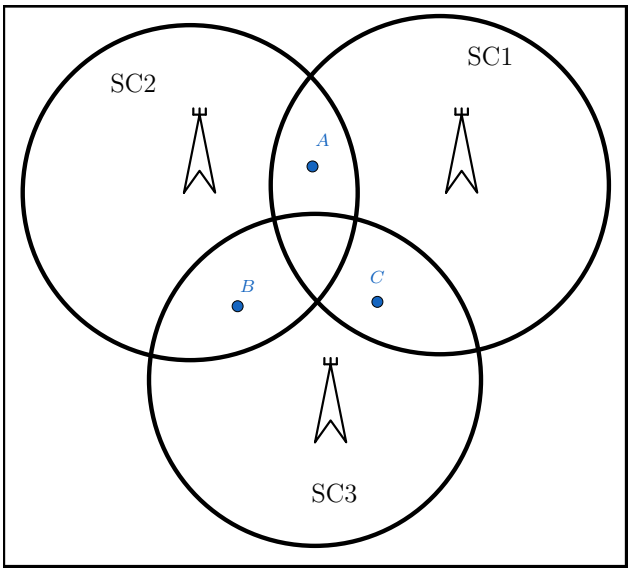

Figure 6.1. Connectivity between users and SCs in a network that consists of three overlapped SCs. Regions A, B, and C are the overlapping area between each two of these SCs that allow users in theses regions to connect to two SCs simultaneously.

RaptorQ mainly contributes to simplifying the problem since the nature of the problem requires in some cases storing a number of distinct encoded symbols that is larger than the file size, E_j, at multiple neighboring SCs. For example, let us consider three SCs overlapping between each other as shown in Fig. 6.1. Let us assume that the optimal distribution for a file, F_j, is to store $\frac{E_j}{2}$ at each SC. Regions A, B and C are the overlapping between each two SCs and users at these regions request this file.

In case that we do not use RaptorQ, users at region A download the first half from SC1 cache and the second half from SC2 cache. Users at region B download the second half from SC2 cache because its already stored there and they have to download the first half

from SC3 cache. At this point, users at regions C can not download the complete file from SC1 and SC3 caches because both two SCs have the first half of the file.

On the other hand, if we use the RaporQ technique, it allows us to generate enough distinct encoded symbols to store distinct encodes symbols of half of the file size, E_j, at each SC. Thus, users at regions C can be served by SC1 and SC3 caches since they have distinct encoded symbols.

The objective function in (6.8) is corressponding to the value of the cache. This value can be maximized by minimizing the amount of downloading on the backhaul link. The cost of downloading file F_j depends on the variable m_j, which is the number of encoded symbols that is placed at the cache from the file F_j.

In order to illustrate the relationship between the value of the cache and m_j, let us consider the case of storing one file of 100 encoded symbols size in the library. This file will be requested by users who are distributed in a network that allows the users to have simultaneous access to up to 4 SCs. The user who has simultaneous access to 4 SCs will prefer to cache 25 encoded symbols at each SC in order to download the complete file from the SC caches. Similarly, the user who has access to three SCs prefers to store the third of the file at each SC. In case we decide to store 25 encoded symbols at each SC, the users who has access to three SC can download 75 encoded symbols directly from the SC caches and the remaining symbols will first be fetched via the backhaul link and then delivered to the user.

The value of the cache, R, is related to the number of the encoded symbols stored in the cache, m_j, $j = 1, 2, ..., N$. Fig. 6.2 shows this relationship. If we store the first 25 encoded symbols, the value of the cache will be about 80 symbols which means the average number of encoded symbols that will be downloaded via the backhaul will be $100 - 80 = 20$ encoded symbols. If we increase the number of encoded symbols to be one-third of E_j, the value of the cache will be about 95 and the average backhauled encoded symbols in this case will be 5 encoded symbols.

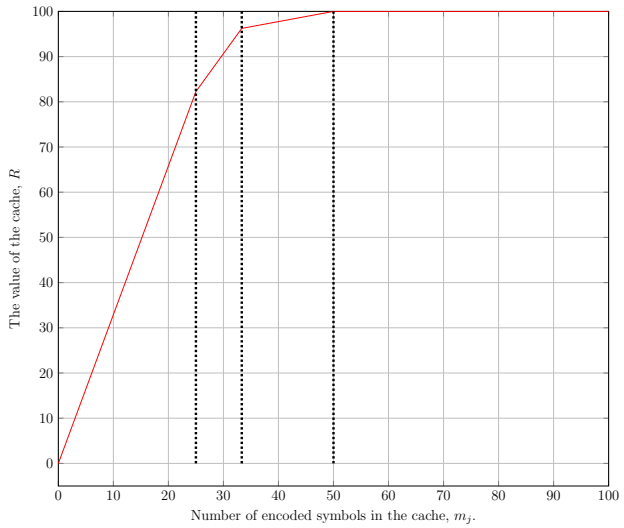

Figure 6.2. The relationship between the number of encoded symbols for some file that is stored in the cache and the value of the cache.

As shown in the figure, the m_j axis is divided into different ranges. The value of the cache for adding symbols is determined by the slope of R over the range that the symbols belong to. Moreover, any two encoded symbols belonging to the same range have the same value of adding to the cache. For example, the value of adding any symbols will be the same among the first 25 symbols. Following (6.8), we can rewrite it as piecewise function which is given by

$$R = \sum_{j=1}^{N} p_j \begin{cases} \gamma_1 m_j + 2\gamma_2 m_j + ... + (S-1)\gamma_{S-1} m_j + S\gamma_S m_j & 0 \leq m_j < \frac{E_j}{S} \\ \gamma_1 m_j + 2\gamma_2 m_j + ... + (S-1)\gamma_{S-1} m_j + S\gamma_S E_j & \frac{E_j}{S} \leq m_j < \frac{E_j}{S-1} \\ . & . \\ . & . \\ . & . \\ \gamma_1 m_j + 2\gamma_2 E_j + ... + (S-1)\gamma_{S-1} E_j + S\gamma_S E_j & \frac{E_j}{2} \leq m_j < E_j \end{cases} \quad (6.12)$$

In the light of this, we have S different ranges, $g \in \{1, 2, ..., S\}$. The first range,

$g = S$, starts at $m_j = 0$ and ends at $m_j = \frac{E_j}{S}$. The second range, $g = S-1$, starts then and ends at $m_j = \frac{E_j}{S-2}$ and so on until the last range ends at $m_j = E_j$. Then, the next step of the analysis is to take the derivative of R respect to m_j, it will then be given as

$$\frac{\partial R}{\partial m_j} = p_j \begin{cases} \sum_{i=1}^{S} i\gamma_i & 0 \leq m_j < \frac{E_j}{S} \\ \sum_{i=1}^{S-1} i\gamma_i & \frac{E_j}{S} \leq m_j < \frac{E_j}{S-1} \\ & . \\ . & . \\ . & . \\ . & . \\ \gamma_1 & \frac{E_j}{2} \leq m_j < E_j \end{cases} \tag{6.13}$$

Now, let us reformulate (6.13)

$$\frac{\partial R}{\partial m_j} = p_j \sum_{i=1}^{g} i\gamma_i \ , \qquad L_g < m_j < U_g \qquad \forall \qquad g \in \{1, 2, ..., S\} \ \ \& \ \ j \in \{1, 2, ..., N\}, \tag{6.14}$$

where

$$L_g = \tfrac{E_j}{g+1} \ \ \forall \ \ g \in \{1, 2, ..., S-1\} \quad \& \quad L_S = 0, \quad U_g = \tfrac{E_j}{g} \ \ \forall \ \ g \in \{1, 2, ..., S\}.$$

From (6.14), we have N files and each file has different ranges where the derivative is constant at each range. The value of the constant depends on the file preference, coverage area probabilities and the range index, g. In case that there are two files with the same preferences, symbols from the corresponding ranges of two files will then have the same value in (6.14), but the number of the available symbols at a certain value of adding to the cache depends on the file size, E_j. For example, If we have two files, F_1 and F_2, with the same preference. File sizes are $E_1 = 400$ and $E_2 = 200$ encoded symbols, $S = 4$. The value of the derivative in (6.14) at the first range of each file will then be the same, $\sum_{i=1}^{4} i\gamma_i$, but

the number of the encoded symbols that is available from each file at this value of derivative will be different. For F_1, the number of encoded symbols will be $\frac{400}{4} = 100$, whereas $\frac{200}{4} = 50$ encoded symbols will be in the case of F_2.

Now, the solution to maximize the objective function, R, is to choose the symbols where the derivative is largest. The optimization problem is to maximize R such that the total number of encoded symbols is M_C. Therefore, the optimal solution is obtained by first calculating all these constant values at each range and then choosing the largest values until the total number of encoded symbols selected is equal to M_C. More detail about the solution is discussed in the next section.

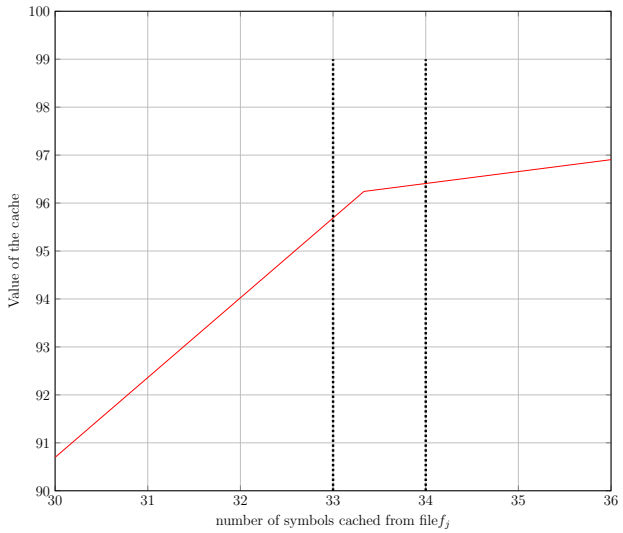

Figure 6.3. Symbol number 34 is fragmented into two parts; each part belongs to a range with different value of caching.

The second constraint in symbols placement problem is that the value of m_j must be integer. For the sake of explanation, let us consider the example of a file that is in the library with a size of 100 encoded symbols. Let us also assume that we store 33 symbols of the file at each SC. The user who has access to three SCs can download 99 symbols and the remaining to reconstruct the file successfully is one symbols, whereas if we store 34 symbols, the SCs have in total 2 symbols more than the user's need. Therefore, the value of 34 in each cache versus 33 is only one symbol. In case the user has access to 2 SCs and each SCs stores 34

symbols, we can then use all 34 symbols at each SC and the symbols 34^{th} is worth 2 symbols. In order to count this, we need to check the upper limit of each range. If the value of U_g is not integer, we then need to consider the symbol where the edge that fragments it into two parts as standalone range with size of one. For example, the edge located in the symbol 34 in the Fig. 6.2 is shown clearly in Fig. 6.3. The symbol 34 consists of two different parts. The value of adding this symbols, 34, can be calculated by determining the summation of two different slopes, which is given by

$$p_j \left(\sum_{i=1}^{g-1} i\gamma_i + \gamma_g (E_j \bmod g) \right). \tag{6.15}$$

Therefore, any file has at least S different values of adding symbol to the cache and $2S-1$ at maximum.

6.3 Maximal Incremental Gain Algorithm

The optimal solution for (6.8) is to choose the symbols where the derivative is largest. The developed algorithm, maximal incremental gain (MIG), calculates first the value of the cache for each range among all files as given in (6.14) and then choose the symbols where the derivative is largest until the cache at the SC is full.

The MIG algorithm, as shown in Algorithm 1, consists of two parts. The first part is to determine all symbol values. The second is to fill the cache with the most valuable symbols. At the first stage, the first for loop is to go over all files in the library. The second loop determines all values of adding file's symbols to the cache at the SC for each file, that is, the value of derivative for each range which is stored in the variable $IncValue$. Each value is stored in the $ValuesList$ array as one entry with the related file index and the number of symbols available at this value which is $IncAvailability$, that is, $ValuesList \leftarrow [IncValue,\ IncAvailability,\ j]$. The loop starts from the last range, g=1, down to the first range, $g = S$. At $g = 1$, the incremental value is $p_j\gamma_1$, and the upper and lower limit is E_j and $\lceil E_j/2 \rceil$ respectively. In the first range, $g = S$, the incremental value is $p_j \sum_{i=1}^{S} i\gamma_i$ and

the upper and the lower limits are $\lceil E_j/S \rceil$ and 0, respectively. At the end of each iteration of the second loop, we check whether the upper limit of the next range is an integer or not. If it is not integer, we then calculate the value of adding the symbol that is located at the edge as in (6.15).

The second parts starts with sorting all entries in $ValuesList$ based on $IncValue$ in descending order. At each iteration in *while* loop, we pick the most valuable symbols from the sorted list, $ValuesList$. $CachSoFar$ represents the cumulative number of encoded symbols in the cache. The value of $CacheCapacity - CacheSoFar$ represents the number of symbols that we can still add to the cache, whereas $ValuesList[x][1]$ is corresponding to $IncAvailability$ which is the number of symbols that are available at the same value, $IncValue$. The minimum operation between these two values, $CacheCapacity - CacheSoFar$ and $ValuesList[x][1]$, is to take care of the case that the value of $IncAvailability$ is larger than the room in the cache. $FractionToAdd$ is the value corresponding to the number of encoded symbols that will be added to cache at each iteration. $SetOfFiles$ is a vector of size N that is updated at each iteration to store the cumulative encoded symbols for the file that we just added symbols from, where the array index is the file number. At the time we break the loop, $SetOfFiles$ is the symbols distribution that gives us the minimum average number of encoded symbols per downloaded file; that is, it is the optimal solution to symbols placement problem.

For the time complexity, MIG algorithm consists of: making the list $ValuesList$, sorting the list, and filling the cache. The part of making the list has a number of operation that is proportional to the number of files, N. The time complexity then is $k_1 N$, where k_1 is constant. The average time complexity of sorting a list in Python is $n \log(n)$, where n is proportional to N. The method that is used in Python for sorting is *Timsort*. Therefore, we can write the average time for this part as $k_2 N \log(k_2 N)$, where k_2 is constant. For the third part, the number of operations is proportional to N since the cache capacity is set to

be $0.05 \times 100 \times N$, that is, $k_3 N$, where k_3 is constant. The whole time can be written as

$$T(N) = k_1 N + k_2 N \log(k_2 N) + k_3 N$$

$$= k_1 N + k_2 N \log(k_2) + k_2 N \log(N) + k_3 N.$$

When we combine the constants, the $T(N)$ will then be as

$$T(N) = c_1 N + c_2 N log(N), \tag{6.16}$$

where c_1 and c_2 are constants. In Chapter 7, we will estimate the value of the these constants from the numerical results.

In summary, the solution of the optimization provides the number of symbols from each file that should be cached at each SC. The amount of data that will then be downloaded is the minimum given the network topology, file preferences, file sizes and number of the files.

In the next chapter, numerical results that illustrate the complexity of the algorithm and the effectiveness of the cache capacity in reducing backhaul usage are presented.

Algorithm 1 Maximal Incremental Gain Algorithm

$ValuesList = [\ ]$
for $j \in \{1, 2, ..., N\}$ **do**
  $IncValue\ =\ 0,\ MaxRange = E_j$
  for $i = 1\ to\ S$ **do**
    $IncValue\ +=\ iP_j\gamma_i$
    if $i\ ==\ S$
      $MinRange\ =\ 0$
    else
      $MinRange = \lceil \frac{E_j}{i+1} \rceil$
    end if
    $IncAvailability = MaxRang\ - MinRange$
    $ValuesList \leftarrow [IncValue,\ IncAvailability,\ j]$
    if $E_j \bmod (i+1) > 0\ \ \&\ \ i\ <\ S$
      $deltaIncValue = IncValue + P_j\gamma_i(Ej \bmod (i+1))$
      $ValuesList \leftarrow [deltaIncValue,\ 1,\ j]$
      $MinRange- = 1$
    end if
    $MaxRange\ =\ MinRange$
  end for
end for

Sort $ValuesList$ in descending order based on $IncValue$
$x\ = 1,\ CacheSoFar\ = 0,\ SetOfFiles = [0, 0, ..., 0]_{1xN}$
while $CacheSoFar < CacheCapacity$ **do**
  $FractionToAdd = min(\ (CacheCapacity - CacheSoFar), ValuesList[x][1])$
  $CacheSoFar = CacheSoFar + FractionToAdd$
  $SetOfFiles[ValuesList[x][2]]+ = FractionToAdd$
  $x+ = 1$
end while

CHAPTER 7

Numerical Results

In this chapter, numerical results are presented to show the the time complexity of MIG algorithm and the effectiveness of the caching in reducing the backhaul usage. Moreover, a compression between the developed algorithm and one of the competitive method to solve such problem is also discussed. All experiments are implemented using Python installed on Intel ®Core™i5 laptop with 8 GB memory.

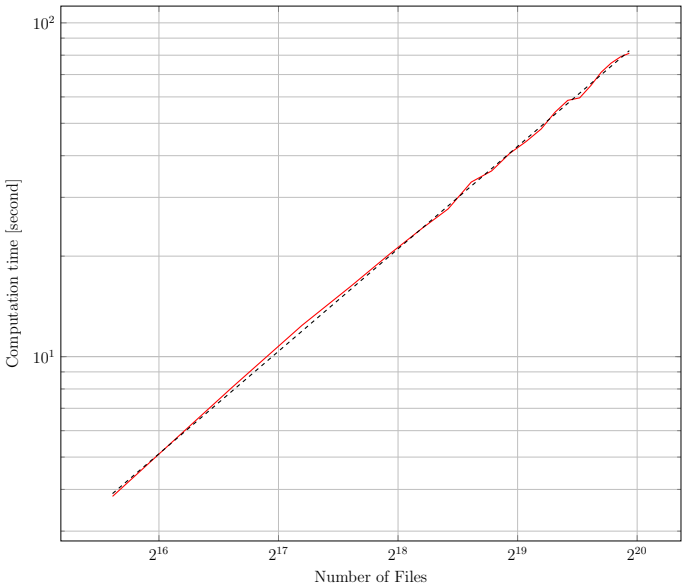

Figure 7.1. The relationship between the the required time by MIG algorithm to find the optimal solution for a library and the number of the files placed in the library. The cache capacity is assumed to be 5% of library size. File preferences and coverage areas probabilities are randomly generated.

Fig. 7.1 shows the time required to find the optimal solution as a function of the number of files in the library. The value of file sizes are randomly generated by developing a routine with a help of *numpy* and *random* modules under the constraint of the library size.

The library size is assumed to 100 MB times the number of files. We also develop a routine for each of file preferences and coverage area probabilities vectors to generate a random value for each one under the constraint the total probabilities of each vector is one. The number of coverage areas is considered 4, $S = 4$. The cache size is 5% of the library size. All of the assumptions that are made in this experiment are chosen arbitrary and will not effect on the correctness of the solution. Any other values could be applicable in order to apply the algorithm on.

We compute the time required to find the optimal solution for different libraries where the range of the number files in theses libraries is between 50,000 to 1,000,000 files with a step size of 50,000 as shown in Fig. 7.1. Black dashed line is the best line fit to the data from (6.16), where $c_1 = 6.043 \times 10^{-5}$ and $c_2 = 3.686 \times 10^{-6}$. Moreover, the time required to find the optimal solution for the library of 1,000,000 files with different sizes and 4 different coverage areas is about 80 seconds, whereas this amount of time is required to find the optimal solution by a competitive method, simplex method, for only few hundreds files as will see in the next experiment.

The second experiment is to show the computations time comparison between MIG algorithm and the simplex method, where we compare the time required to find the optimal solution between them for different libraries.

The simplex method is considered efficient method in practice to solve linear programming problems. The concept behind simplex method is that each inequality constraint defines as a hyperplane and a feasible half-space. The intersection region of all feasible half spaces is called the feasible region. The simplex method follows an itrative procedure by moving from one corner to another until the optimal solution is found, where the optimal solution to minimize or maximize the objective function subject to the constraints is located at one of feasible region corners. Even though the simplex method follow this procedure to find the optimal solution, it dominates the filed of linear programming. However, the average time complexity is difficult to analyze and it depends in particular on the distribution of the

problem. On the other hand, *Klee* and *Minty* [5] presented an example showing that the worst-case complexity for simplex method is exponential in the number of constraints.

Another point that must be mentioned is that the solution found by the simplex method for our specific problem is slightly different form the optimal solution. The reason for this is that the simplex method can not consider the integer constraint for the number of encoded symbols in its constraints. Thus, the solution would include a fractional number of encoded symbols. As explained before, this constraint is important since the number of encoded symbols can not be fractional. However, if we modify our algorithm to not consider the integer constraints, the solution of our algorithm and the simplex method are identical. In order to consider the integer constraint, we need to solve the problem by one of integer programming methods which has higher complexity than the simplex method. In contrast, the complexity of MIG algorithm is $n \log n$ because it is a list sort.

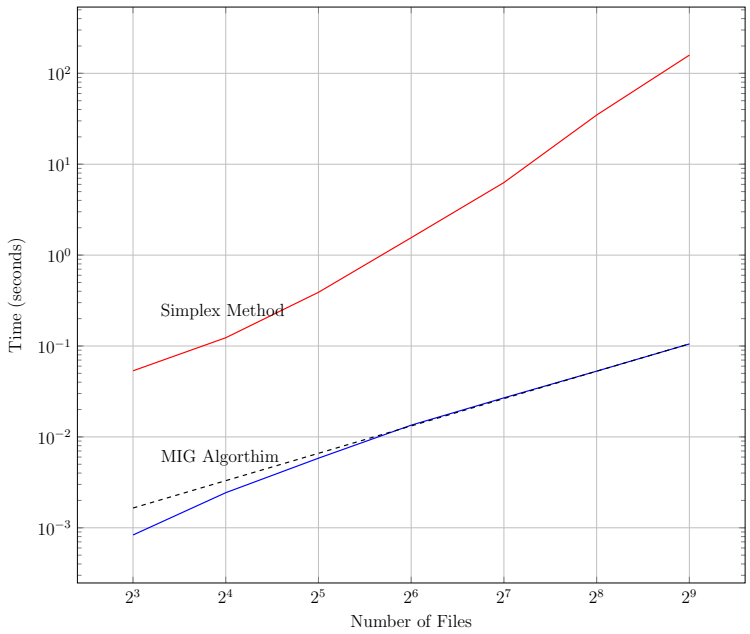

Figure 7.2. A comparison of the required time to find the optimal symbols distribution at different library sizes for MIG algorithm and simplex method. MIG algorithm significantly outperforms the simplex method. The required time gap between them increases as the number of the files in the library increases.

It is clear from the figure that there is a significant difference between two methods,

where MIG algorithm greatly outperforms the simplex method. Moreover, as the number of files increases the time difference increases. The reason for this is that the simplex method complexity depends on the number of corners in the feasible region which are proportional to the number of files, whereas MIG algorithm has $n \log n$ time complexity where n is the number of files. Moreover, we can not do the optimization for more than 700 files using the simplex method because the memory requirements exceed the available memory.

The calculations that is done by MIG algorithm to find the optimal solution is repeated 100 times. The result that is shown in Fig. 7.2 is the average time of all 100 iterations. The reason for this is that the required time to find the optimal solution is a fraction of second for small library of files. Thus, the average time of 100 iterations will be more accurate than considering one iteration.

In this experiment, file sizes, file preference, and coverage area probabilities are randomly generated as the same as the first experiment. The library size is assumed to be 100 MB $\times N$. Cache capacity is also assumed to be 5% of the library size. All the assumptions are chosen arbitrary and do not affect on the correctness of the solution.

In order to evaluate the effectiveness of the cache capacity in reducing the backhaul usage, we applied MIG algorithm on a YouTube requests trace data to determine the average size of backhualed encoded symbols per user request as a function of the cache capacity.

The related trace data was collected on the Amherst campus of youtube vedio requests in 2008 [8]. It consists of about 250 thousand distinct videos and about 600 thousand requests. The trace data was collected over a period of 3 days. The size of the videos is not mentioned in the trace. Thus, we tried to fetch the sizes with help of *pafy* library. *pafy* is a Python library to download YouTube content and retrieve metadata and it is named as *pafy* with no *py* at the end. Even though we use this library, we could not have this information for two reasons; the time required to fetch the sizes of first 3 thousand videos takes 3 hours, that is, the total required hours to find the size for all videos is too large and 60% of these 3,000 videos is no longer available since the trace was done in 2008, that is, more than half

of the trace is no longer available. In light of this, we assumed that all videos has the same size which is 30 MB. This size is reasonable for a video of around 3 minutes playback time and screen size of 640 x 360.

The topology of the network that is considered in this experiment is shown in Fig. 7.3, where the radius of SC is 60 meters and the vertical and horizontal distance between any two SCs is 60 meters. Each arch represents a part of one coverage area of SCs neighboring the central SC. Users are uniformly distributed. Thus, the coverage area probabilities can be calcuated from (6.11), where $\gamma = (0\ ,\ 0.11,\ 0.49,\ 0.40)$.

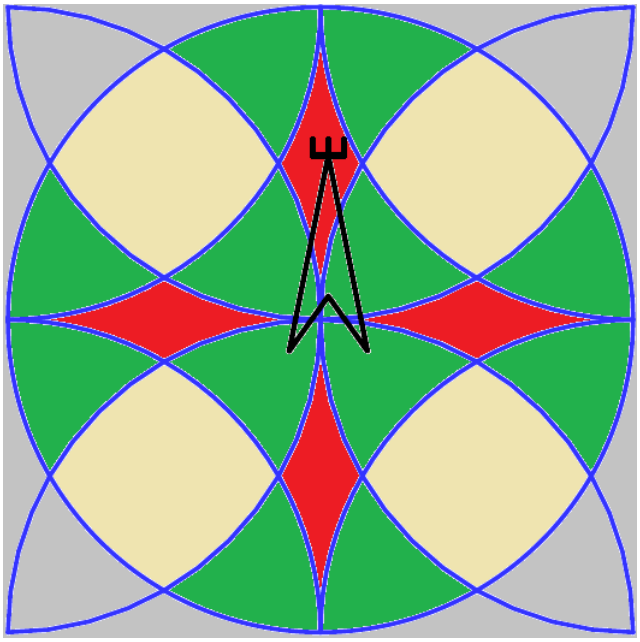

Figure 7.3. Coverage areas in a SC with radius of 60 meters, where the vertical and horizontal distance from neighboring SCs is 60 meters. Red, Green and Paige are the areas where the user has simultaneous access to 2, 3 and 4 SCs, respectively.

The values of the file sizes and the network topology are chosen arbitrary and any other values or different network topology will not effect in the correctness of the solution. The result of this experiment is shown in Fig. 7.4. The figure shows the average size of backhauled encoded symbols per user request as a function the cache capacity. The cache size is in a range of 0 and 30 GB with a step of 0.5 GB. As shown in the figure, the average

size of backhauled encoded symbols is 30 MB with no cache at all. This value is reasonable since its equal to the file size itself. The average size then decreases down to about 25 MB per user request with caching only 30 GB from the library. That is, the decreasing percentage for average number of encoded symbols per downloaded file is 16.6% with caching only 0.379% of library size. The average time required to compute the optimal symbols distribution at each different cache size value is 26.6 seconds.

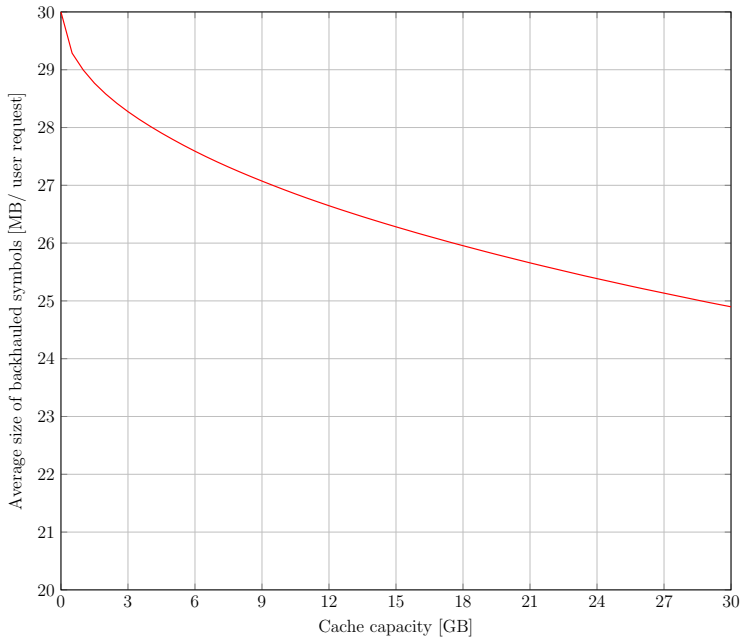

Figure 7.4. The effectiveness of the cache capacity at the SC in reducing the average size of backhaulded encoded symbols per user request. The average size decreases as the cache capacity increases.

The last experiment shows the effectiveness of cache capacity in reducing the data rate in the backhaul link. To determine the effectiveness, we applied the algorithm on a small set of 50 YouTube videos trending on October 20, 2017. The related metadata to this set is shown in Table 7.1, where the value of m_j represents the optimal symbols distribution at the SC when the cache size is 1 GB . The preference for a given video is calculated as the number of views for this video divided by total number of views for all videos. The total number of views is 50,370,554. The average videos size for the set is about 60 MB. The total

set size is about 3 GB. The topology of the SC network is the same as SC in Fig. 7.3, that is, $\gamma = (0\ ,\ 0.11,\ 0.49,\ 0.40)$. Symbols size is assumed to be 1 kB.

Fig. 7.5 shows the average data rate in the backhual link as a function of cache size for three different requested rates: $10^2/hour$, $10^3/hour$ and $10^4/hour$. From the figure, it is clear that when there is no file stored at the SC, that is, cache capacity is zero, the data rate is at the maximum value. The data rate decreases significantly in the three cases as the cache size increases. When the cache size is equal to one GB, that is, one third of the total set size, the date rate will be almost zero in the case of 10^4 $request/hour$.

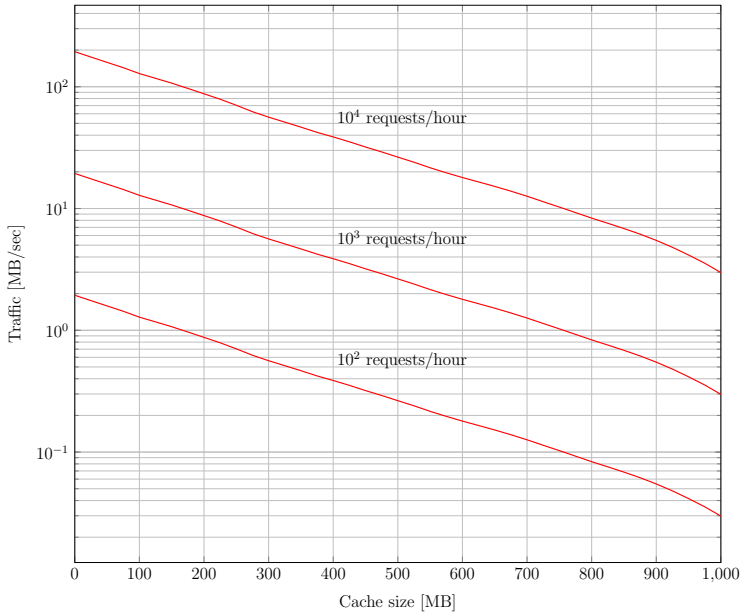

Figure 7.5. The effectiveness of cache capacity at the SC in reducing date rate on the backhaul link. The effectiveness is measured for three different request rates: $10^2/hour$, $10^3/hour$ and $10^4/hour$. The date rate decreases as the cache capacity increases.

Table 7.1. Metadata for 50 YouTube videos trending on October 20, 2017.

No.	YouTube Video ID	File size (kB)	preferences	m_j
1	EUoe7cf0HYw	32051	0.0827	16026
2	zJqnpoJPDMM	77746	0.0815	38873

3	AQ0b0CRsiVc	55221	0.07419	27610
4	CwfoyVa980U	20839	0.07296	10419
5	_95NxbsbMF4	215428	0.06618	107714
6	WHvT-oGf6ik	17472	0.05258	8736
7	5PJWCe92K1E	31145	0.05098	15572
8	9wRgNPdU7fQ	119082	0.04466	59541
9	fAuUd1HiWe4	141018	0.04454	70509
10	ketCUtFpzNo	30738	0.04057	15369
11	7KbjNfCV-lY	213035	0.03731	106517
12	-Denciie5oA	24360	0.03722	11605
13	OMW_dPtm7Bo	17411	0.02824	5804
14	vGVK4ibMI-Y	32359	0.02624	10787
15	M2Kwpy2ot6k	4395	0.02248	1465
16	PKdKzs0xS9c	133581	0.02027	44527
17	g9oCOSvYkWc	12115	0.01756	4039
18	c_weoC3HT6Q	5225	0.01643	1742
19	xtsLwr9Ok7s	128006	0.01517	42669
20	WPlx5BmZVkQ	49110	0.01486	16370
21	Upqp21Dm5vg	119669	0.0145	39890
22	a_MFkqMZTt4	170727	0.01197	56909
23	crY-q80MfJU	120752	0.01067	40251
24	GoJsr4IwCm4	24152	0.01063	8051
25	H1–Pifm05Q	22386	0.01011	7462
26	xRQGzS6Ce-g	17941	0.00915	5980

27	qAPx7N1jr_I	5569	0.009	1856
28	IZ5izPbcgG4	15360	0.00846	5120
29	C25qzDhGLx8	19908	0.00844	6636
30	6cnobaJFY-M	20716	0.00727	6905
31	GjtYtBGrP6Y	42483	0.00661	14161
32	yl86_6Rr-mU	67380	0.00594	22460
33	42G_XzaPcPo	165111	0.00573	55037
34	pcnlLwwR5x4	59323	0.00538	19774
35	lNjwsz8bLLI	25181	0.00445	6296
36	rdXaw4EFGdk	43516	0.0036	10879
37	FxElTWf7IEw	31817	0.00345	7954
38	EWng8GQ49Fg	93571	0.00319	23393
39	ksdAs4LBRq8	60200	0.00303	15050
40	yESCuF5dsYs	120170	0.00285	30042
41	wVD8ECqG2FM	27131	0.00214	0
42	upd8nVJr7lc	56046	0.00214	0
43	b1wlmTNERqE	10972	0.0017	0
44	TQKWC37ful4	6705	0.00096	0
45	7xCQ4B40wOc	52315	0.00086	0
46	SmW6p1HwNtw	28544	0.00038	0
47	kAphgHhlteM	43241	0.00031	0
48	Z9KnnXPoP_o	4098	0.00024	0
49	5otKiSs3AeY	112153	0.00014	0
50	sQTnREEtuNk	44036	7e−05	0

CHAPTER 8

Conclusions

In this study, we addressed the symbols placement problem, for which the problem formulation and the optimal solution are presented. The maximal incremental gain (MIG) algorithm is developed for this purpose. The caching strategy takes advantage of RaptorQ properties. The algorithm determines the optimal number of encoded symbols from each file in a library that be should placed at small cell SC caches. The value of the cache at these SCs will then be at the maximum value, which is obtained by minimizing the number of backhauled encoded symbols. The algorithm has a low time complexity–$n \log n$, where n is the number of files in the library–and considers multiple variables in its calculations: cache capacity at the small cell, file sizes and preferences, the number of files in the library, and the network topology.

Numerical results that illustrate the complexity of the algorithm and the effectiveness of the cache capacity in reducing backhaul usage are presented. We measured the time required to find the optimal symbols distribution as a function of number of the files in the library. We compared computation time for the MIG algorithm to that of the simplex method. We also evaluated the effectiveness of cache capacity in reducing backhaul.

One of the interesting results is that the time to find the the optimal solution for a library with 1,000,000 files is about 80 seconds, whereas the simplex method requires amount approximately of the same time to do the calculations for a library of 500 files. Moreover, our result shows the effectiveness of the cache at the SC in reducing the backhaul link usage in terms of the average size of backhauled encoded symbols per downloaded file and the average data rate on the backhaul link. Our results show that caching a fraction of 1% (0.379%)

of a set of files is enough to reduce the average size of the backhauled encoded symbols per download file by 16.6%. In other experiments, caching 10% of a set of files is enough to reduce the data rate of the backhaul link by 65%.

To the best of our knowledge, an efficient protocol for coordinating download of files that are stored as a distributed collection of encoded symbols has not been presented in the literature. The development and standardization of such an algorithm would appear to be a worthwhile direction for future work.

Because of the nature of the RaptorQ encoding system download of file can begin and end with any encoded symbols. Thus, for example, multiple downloaders cloud collect symbols from cell site transmissions starting at any point in a file download. The impact of such multiple reception would appear worth-well investigating.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] (2014), Urban Small Cell Network Architectures, *Tech. Rep. SCF088*, Small Cells Forums.

[2] (2014), Extensions of small cell zone API, *Tech. Rep. SCF093*, Small Cells Forums.

[3] Bioglio, V., F. Gabry, and I. Land (2015), Optimizing MDS codes for caching at the edge, in *Proceedings of 2015 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, doi:10.1109/GLOCOM.2015.7417697.

[4] Golrezaei, N., K. Shanmugam, A. G. Dimakis, A. F. Molisch, and G. Caire (2012), Femtocaching: Wireless video content delivery through distributed caching helpers, in *Proceedings of IEEE INFOCOM 2012*, pp. 1107–1115, doi:10.1109/INFCOM.2012.6195469.

[5] Klee, V., and G. Minty (1970), *How good is the simplex algorthim.*, Defense Technical Information Center.

[6] Luby, M., A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder (2011), RaptorQ forward error correction scheme for object delivery. IETF request for comments, *Tech. Rep. RFC6330*.

[7] Rouayheb, S. E., A. Sprintson, and C. Georghiades (2010), On the index coding problem and its relation to network coding and matroid theory, *IEEE Transactions on Information Theory*, 56(7), 3187–3195, doi:10.1109/TIT.2010.2048502.

[8] Zink, M., K. Suh, Y. Gu, and J. Kurose (2009), Characteristics of YouTube network traffic at a campus network - measurements, models, and implications, *Computer Networks*, 53(4), 501–514.

VITA

Ibrahim Freewan

401 Hathorn Road · Oxford, MS 38655 · (937) 838 - 2880 · ibrahim.freewan@gmail.com

EDUCATION

M.S., Electrical Engineering, University of Mississippi, May 2018

Thesis: Maximizing Cache Value for Distributing Content via Small Cells in 5G

B.Sc., Electrical Engineering, Yarmouk University, February 2014

Graduate Project: Multi band Micro-strip Antenna Design

EXPERIENCE

Teaching Assistant, 2015 - 2017

University of Mississippi

Courses: Local Area Network, Electronics

Technical Consultant, 2014 - 2015

DIMA Solutions Company